Методичка 5.2 - Создание ключа

Поиск и анализ функции валидации

Давайте откроем программу prog-4.1 под отладчиком.

Первое, с чего стоит начать, это найти функцию, которая осуществляет проверку ключа. Это можно сделать двумя способами:

1. Проследить весь путь работы с ключом с самого начала программы;
2. Найти функцию по ключевым строкам.

Давайте в отдельном окне запустим программу и попробуем активировать.

> prog-4.1.exe

Usage:

prog-4.1.exe <number> <number>

Activation:

prog-4.1.exe <license key>

Из подсказок программы мы узнаем, что ключ должен передаваться через входные аргументы при запуске программы, значит ключ попадает в один из аргументов функции main, который называется argv. Входной аргумент argv является массивом. Первый элемент этого массива это всегда название программы, второй аргумент это первый параметр, который был передан при запуске программы. Значит ключ будет находиться в argv[1].

Давайте попробуем активировать программу.

> prog-4.1.exe blabla

License key is incorrect.

Видим, что программа выдает сообщение “License key is incorrect.”, если ключ неправильный.

Давайте попробуем найти функцию, в которой используется строка “License key is incorrect.”.

Переходим в начало секции кода и ищем ссылки на нужную нам строку.

Видим, что первая функция, это usage, она вызывается, когда мы запускаем программу без входных аргументов. Идем дальше.

Находим строку в одной из функций. Это строка передается в качестве единственного аргумента в функцию 0x00401690.

Как можно догадаться, это сообщение мы видим, когда ключ уже был проверен. И здесь нас просто уведомляют о том, что ключ неправильный. Сама проверка ключа должна идти раньше.

Очень похоже на то, что функция 0x00401690 - это функция printf. Можем поставим точку останова перед вызовом этой функции и запустить программу с одним параметром.

Мы остановились перед вызовом функции. Видим, что пока в командной строке не было записей.

Выполняем вызов функции. Видим, что запись появилась, значит это функция printf. Добавим комментарий и запомним, что 0x00401690 - это функция printf.

Если посмотреть на код, который идет дальше, то можно увидеть ссылку на строку “The program has been activated!”. Но мы туда не попадем, так как далее нас ждет безусловный переход JMP.

Давайте обратимся к коду, который идет перед тем, как будет напечатана строка “License key is incorrect.”. Мы видим вызов функции 0x00401050 и условный переход. Видим, что переход JNZ указывает на код, где должна быть напечатана нужная нам строка.

Давайте разберемся, от чего зависит результат условия. После вызова функции происходит запись значения из регистра AL в регистр ECX. Далее мы видим оператор TEST, который выполняет проверка на нулевое значение в регистре ECX. И прыжок будет, если результат будет не 0.

Мы выяснили, что решение о корректности ключа принимает функция 00401050. Давайте посмотрим на содержимое этой функции.

Кликаем правой кнопкой на инструкции CALL и выбираем Follow.

Это, предположительно, функция, которая выполняет проверку ключа. Давайте выполним анализ этой функции.

В самом начале мы видим пролог, затем сохранение значения регистра ECX в стеке.

Давайте поставим точку останова перед инструкцией MOV и перезапустим программу.

Выполняем инструкцию MOV и видим, что адрес в регистре EAX указывает на ключ, который мы передали при запуске программы.

Вызываем эту функцию и видим, что в регистре EAX появилось значение 6. Это значение похоже на длину нашей строки. Давайте изменим длину ключа (blabla1) и перезапустим программу.

Видим, что теперь в регистре EAX значение 7, значит функция с адресом 0x004036E0 это функция strlen. Добавим комментарий.

Далее мы видим, что длина сравнивается со значением 0x13 (19). Мы эту проверку не проходим и попадаем в конец функции и функция завершается со значение 0 в регистре EAX. А мы помним, что нам необходимо добиться того, чтобы функция вернула ненулевое значение.

Ставим ключ с длиной в 19 символов и перезапускаем программу. Мы прошли первую проверку, теперь мы знаем, что ключ должен состоять из 19 символов.

Проверяем ключ в командной строке.

> prog-4.1.exe 1234567890123456789

License key is incorrect.

Ключ всё ещё неправильный. Продолжаем анализ функции.

Далее в регистр ECX помещается значение 1, затем за счет сдвига битов влево на 2 позиции мы получаем значение 4 в регистре ECX. Затем в регистр EDX попадает указатель на ключ. Далее в регистр EAX попадает значение, который находится в строке на 4-й позиции. Это байт 35, который по ASCII таблице равен 5.

И происходит сравнение с байтом 0x2D. И мы попадаем в конец функции с нулевым значением в регистре EAX. Делаем выводы, что 4-й байт должен быть 0x2D. По таблице ASCII это знак “-”.

Изменяем ключ (1234-67890123456789) и перезапускаем программу.

Мы прошли проверку и далее видим еще несколько таких же проверок. На 9-й и 14-й байты.

Проверяем ключ в командной строке.

> prog-4.1.exe 1234-6789-1234-6789

License key is incorrect.

Ключ всё ещё неправильный. Продолжаем анализ функции.

Изменяем ключ (1234-6789-1234-6789) и перезапускаем программу.

Далее мы видим набор инструкций, который очень похож на цикл с предусловием.

Выставляет локальная переменная в 0. Затем сравнивается с 0x14 (20). Это длина нашего ключа.

Далее мы видим условный переход JGE, который прыгает на инструкцию MOV, которая кладет значение 1 в регистр AL и функция завершает свою работу. Это то, что нам нужно.

Получается, нам нужно оставаться в цикле пока счетчик не достигнет 20.

Итак, мы попали на первую итерацию цикла. В регистр ECX мы кладем указатель на ключ. Затем прибавляем счетчик. Видим, это смещение по строке. Первый байт ключа кладем в регистр EAX. Прибавляем счетчик. Затем в регистр ECX кладем второй байт ключа. В регистре EDX храним сумму первых двух байтов. В регистр ECX кладем третий байт и снова прибавляем его к значение в регистре EDX. Затем тоже самое с третьим байтом. Далее мы кладем четвертый байт в регистр ECX. И теперь с помощью инструкции LEA мы складываем значение в регистре EDX и ECX и вычитаем число 0xC0. В результате получаем число 0x0A (10) в регистре EDX и выполняем сравнение с 0x0A. Они равны и происходит переход к началу цикла. Получается, мы посчитали сумму первых 4 байт и вычли из суммы 192.

Переходим ко второй фазе цикла. Мы увеличиваем счетчик на 5 и на этот раз складываем следующие 4 числа в ключе. В сумме вы получили 1E и вышли из цикла со значением 0 в регистре AL.

Получается, что цикл с каждым проходом считает сумму цифр в каждом блоке, вычитает 192 и сравнивает с 10. Первый блок прошел проверку, а второй уже нет. Нам ничего не мешает сделать второй блок и последующие таким же, как первый.

Проверяем ключ в командной строке.

```
>prog-4.1.exe 1234-1234-1234-1234

The program has been activated!
```

В этот раз ключ прошел проверку. Проверка была основана на сумме байт в каждом блоке ключа. Сумма каждого блока должна быть равно 10.

ASCII код цифры 1 - это 0x31 (49), цифры 2 - это 0x32 (50) и т.д. Для того, чтобы получить число, нужно вычесть 48 из каждого байта. Например, 49 - 48 - это 1, 50 - 48 - это 2 и .т.д. Если

мы складываем 4 числа в каждом блоке, то в конце нужно вычесть 4 раза по 48. А это и есть 192, которые мы видели в коде

Давайте составим другой ключ, но с той же суммой в блоке.

> prog-4.1.exe 0334-1234-1234-1234

The program has been activated!

>prog-4.1.exe 0334-1333-2224-0244

The program has been activated!

Думаю, алгоритм проверки ключа понятен. Проверим работу программы.

> prog-4.1.exe 1 2

Result: 3

Как мы видим, программа работает и уже не выводит информации о триальном периоде.

Алгоритм проверки ключа

Мы выполнили реверс-инжиниринг программы, которая уже очень близко приближена к настоящим программам, который вы можете встретить на своем компьютере.

Мы восстановили алгоритм проверки ключа, который состоит из трех этапов:

- Проверка длины ключа. Ключ должен быть ровно 19 байт.
- Ключи должен иметь определенную структуру. 4 блока по 4 байта, которые разделены знаком тире.
- Сумма цифр в каждом блоке должна быть равно 10.

Если все эти условия будут выполнены, то ключ пройдет проверку и активирует программу.